San Jose State University

Project Report on,

Simulation of Reddit Using Distributed Enterprise Technologies

Under guidance of,
Prof. Simon Shim

CMPE 273 Enterprise Distributed Systems
Spring, 2021

:Team 02:

Alihussain Ladiwala,
Danesh Vijay Dhamejani,
Geetika Kapil,
Karan Pinakinbhai Jariwala
Sai Nikhil Yandamuri
Yusuf Juzar Soni

Contributions:

- **Alihussain Ladiwala**
    - Implemented Redis Client for the application, we are caching the messages for fetching products. If the product search query is present in the cache, then the results are fetched from the cache.
    - Contributed to DB Design, UI Design
    - Created myCommunity Page
    - Contributed to UI utils such as redux, routing, code structuring for the UI
    - Created Account Page
    - Assisted with the chat functionality
    - Assisted with Moderation Page, List communities Page
    - Assisted with Deployment
    - Assisted with Kafka setup for the project.

- **Danesh Vijay Dhamejani**
  Implemented the product search and filter, product view for individual product
    - Implemented API's for Analytics page.
    - Implemented all API's pertaining to SQL.
    - Created MongoDB and SQL models.
    - Implemented multiple image functionalities
    - Implemented backend testing using Chai and Mocha.
- **Geetika kapil**
    - Implemented search community page
    - Implemented moderation page
    - Added sorting features in my community page
    - Application deployment
    - Assisted with Deployment
    - Assisted with Kafka setup for the project.
    - Worked extensively to integrate the above mentioned UI components with their respective backend components.
    - Report

- **Karan Pinakinbhai Jariwala**
    - Implemented all API's for Post in reddit.
    - Implemented all API's for Comments in reddit.

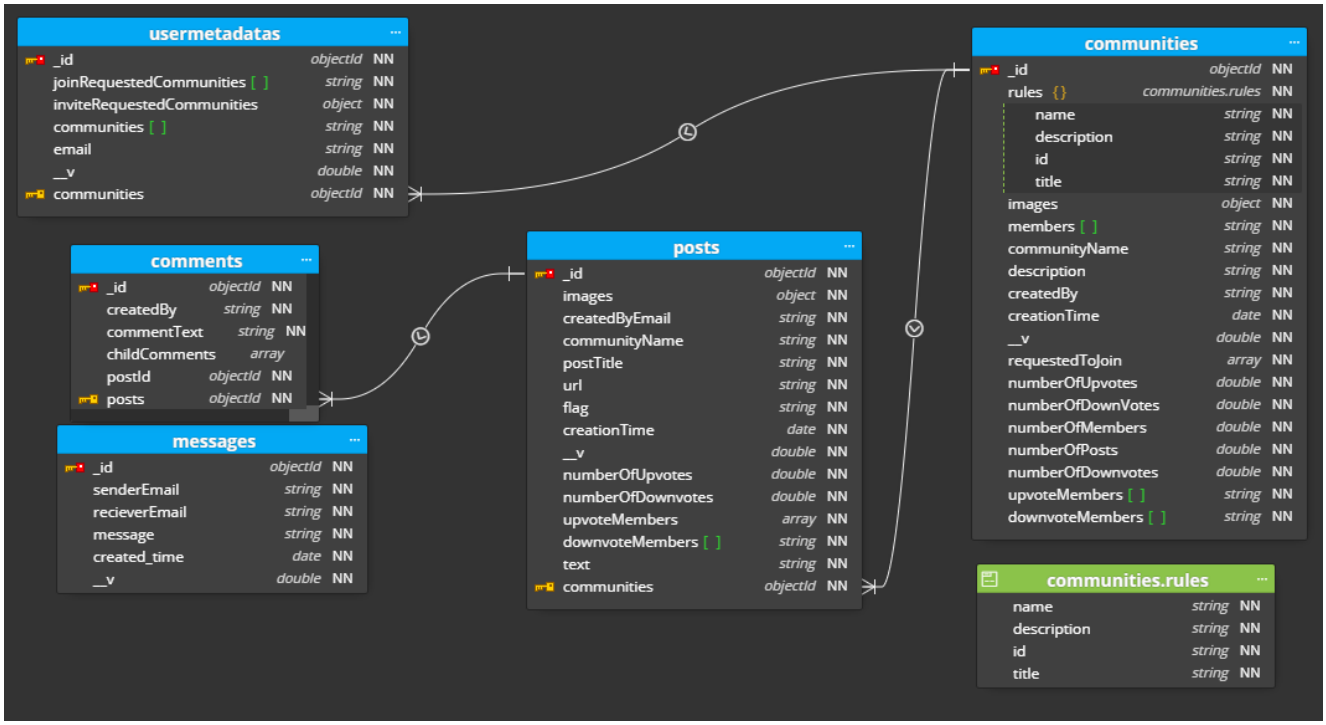
- Setup initial Kafka requirements.
- Implemented API's for message.
- Created Analytics page with graphs implementations.
- Assisted in writing other API's and some bug fixes in the final iteration.
- Backend API Testing in chai,mocha

- **Sai Nikhil Yandamuri**
  - Chat with other users functionality was implemented with usage of socket.io.
  - The Invitation Acceptance page was implemented in UI and backend.
  - Was integral in designing the database and security management throughout the application.
  - Assisted with deployment.
  - Developed API's required for My Communities and Search Community pages.
  - Developed API's required for Messages in chat application.
  - Assisted in writing other multiple API's.

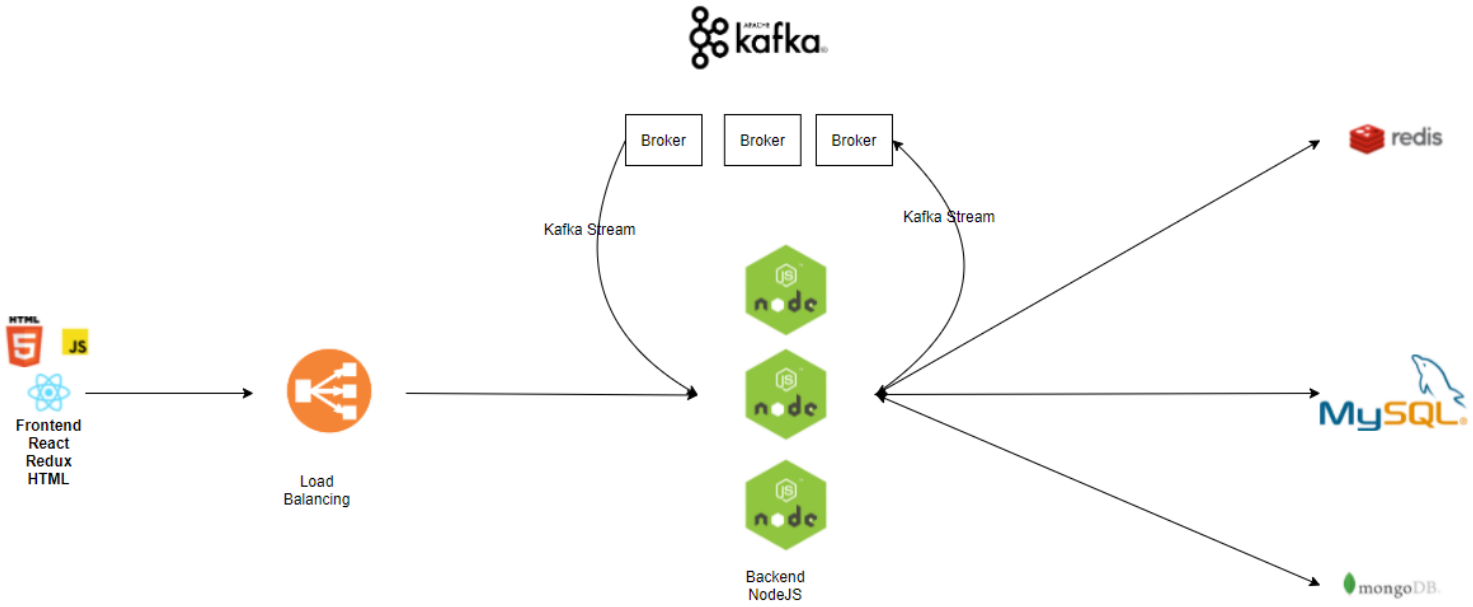
- **Yusuf Juzar Soni**

  - Took active part in discussions during the Database and Architecture design phase of the project.
  - Helped interpret and identify key functionalities from given requirement documents and brainstormed on possible ways to implement the same.
  - Developed the UI for the Home (Dashboard Page) of the application using a combination Material UI, React Bootstrap and HTML 5 technologies.
  - Developed the UI for Create Post and View Post using the above mentioned technologies.
  - Developed UI for the Comments components in the View Post Page
  - Worked extensively to integrate the above mentioned UI components with their respective backend components.
  - Carried out extensive manual testing to validate and verify whether all assigned components were working smoothly.
  - Helped with debugging of minor errors and corner cases.
  - Tried to implement input validation in all components assigned for development.

Database Schema:



System Architecture Design diagram:



Notes:

Your object management policy:

Backend

- The dynamic objects like comments,messages,posts,communities etc are stored in mongodb as they are used frequently. The user credentials are stored in MySQL along with the user topics.
- MongoDB fetches the data with higher throughput making it feasible to do read and write operations on a large amount of data. These objects require scaling.
- The most sensitive information of the user is stored in MySql.
- Large images are stored on the Amazon S3 Bucket. For data retrieval, these images are stored on different cloud instances

Frontend:

- **Reusability :** By using functional components and props we were able to reuse The data fetched from the backend.
- **Redux:** To make the information readily available across application redux store were used. It was used to store login data for our application. It provided the feature of plug and play so that it is easy to use hooks for fetching the state of objects.

Heavyweight Resource Handling:

Connection Pooling:

- To achieve better system performance, we are using connection pooling. We used it as it makes it easy for us when application scales up and it helps to increase system performance.

Redis Caching:

- For faster retrieval of the messages from the database we used redis cache. It aids to optimize application performance.

Denormalized Data in MongoDB:

- MongoDB helped us to execute queries on the database without using complex joins. It helps us when

Kafka Messaging Queue:

- Real time messaging systems were possible due to kafka. It helped when the number of posts created by the user increased. It provide fault tolerance and scalability to our project

Database write policies:

MySQL:

- For data protection, we have stored the personal data of users in MySQL.
  It helps to structure our data.

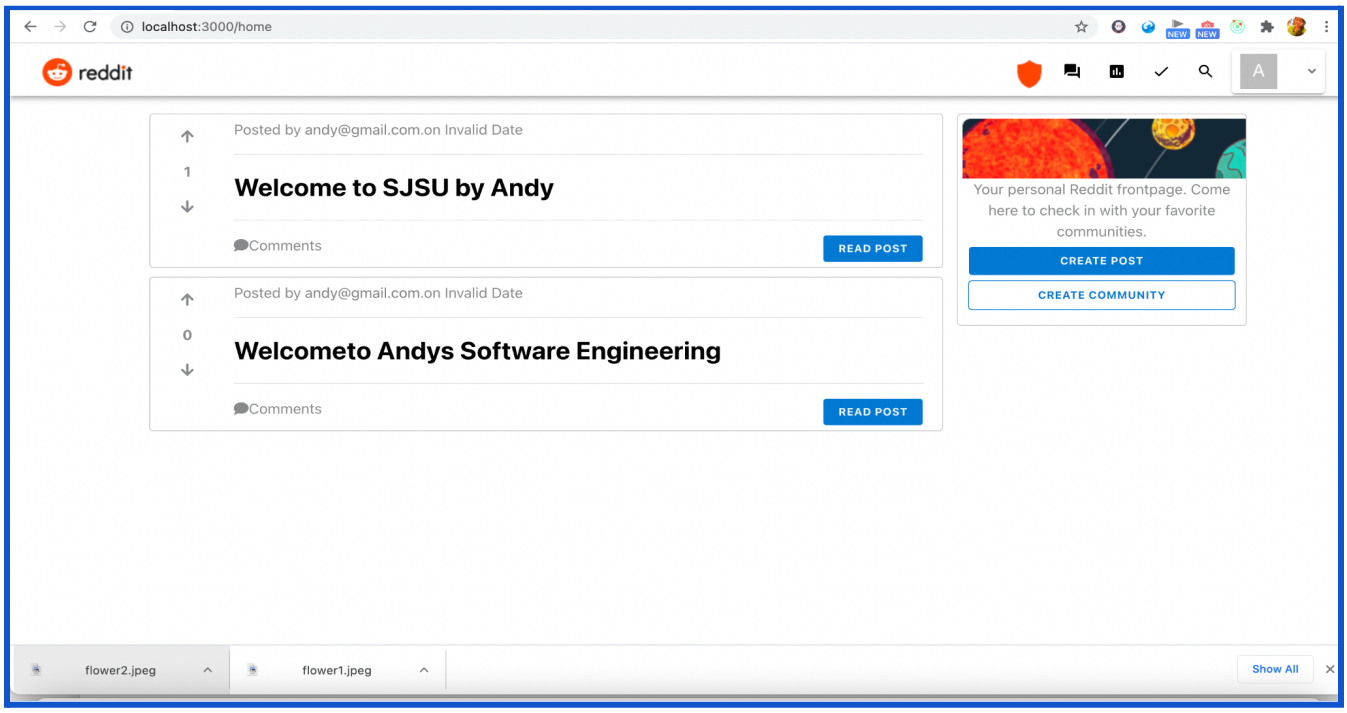
MongoDB:

- As mongodb is schema less database its structure is not defined. We are storing posts,comments, messages,communities in Mongodb for faster retrieval

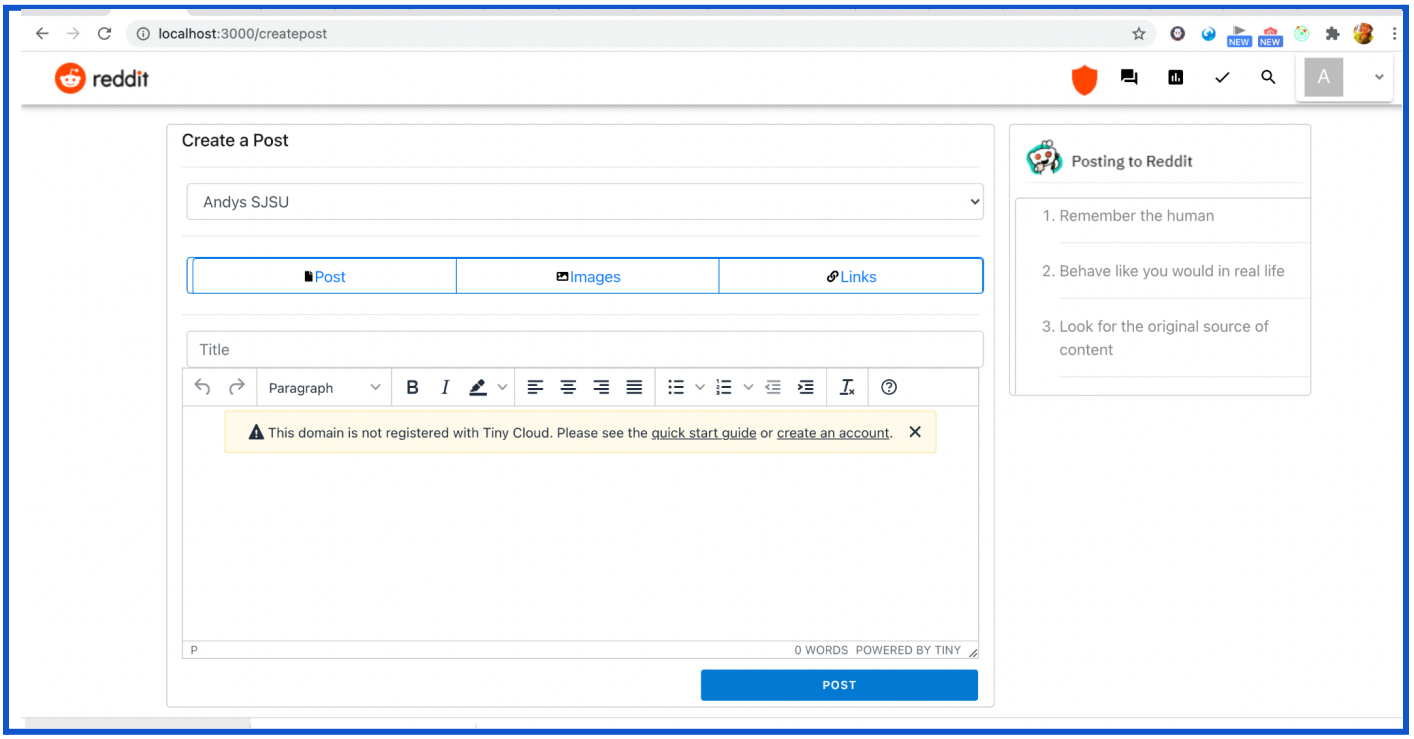
- Auto sharding and scaling helps to increase the performance of the application.

Screen Captures of Application:

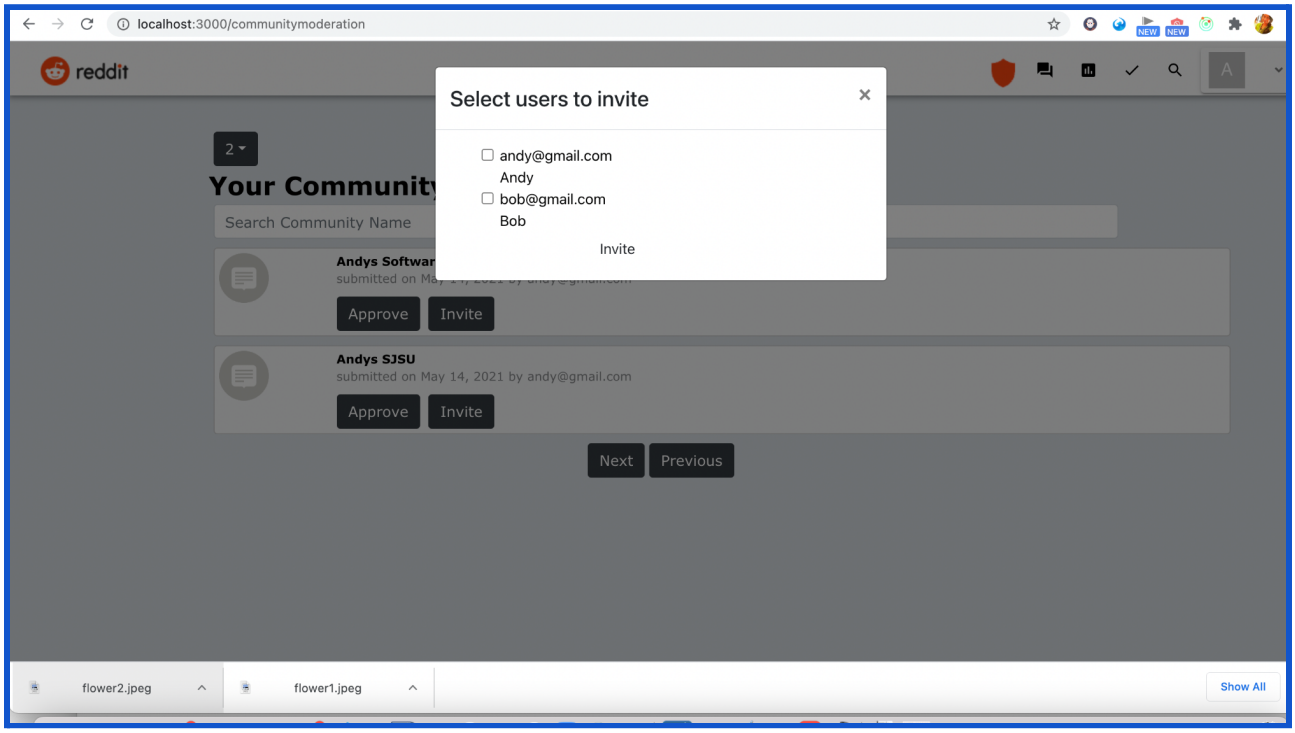
Home Page:

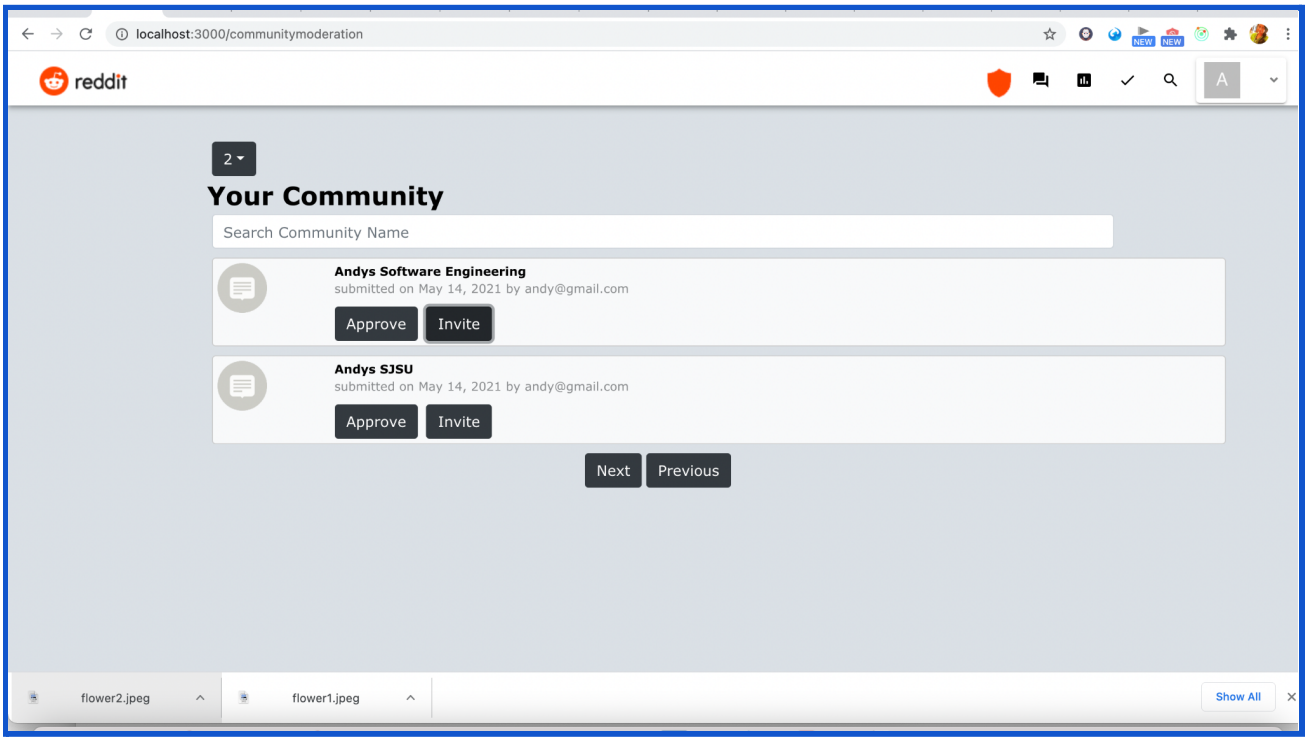


Create Posts

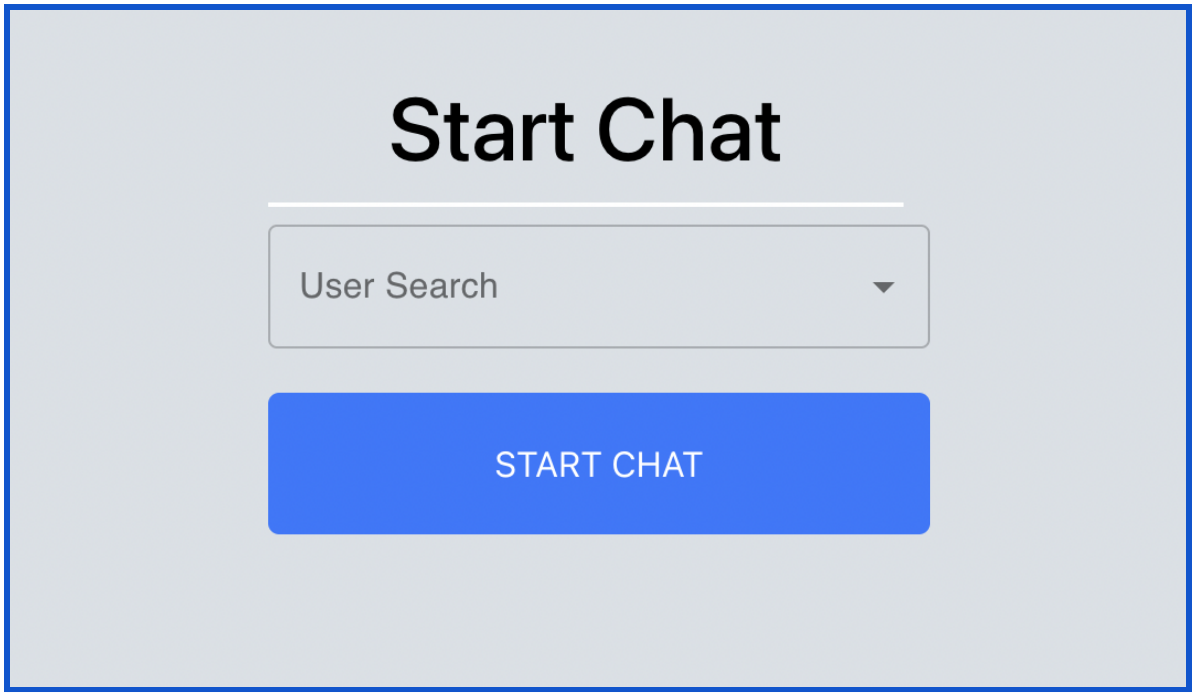


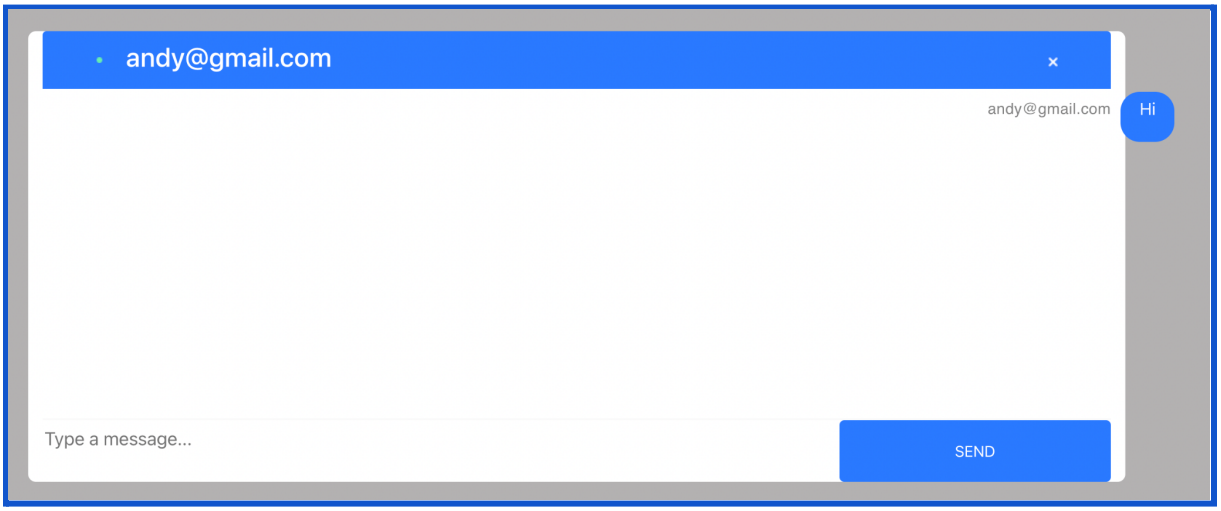
Community moderation page



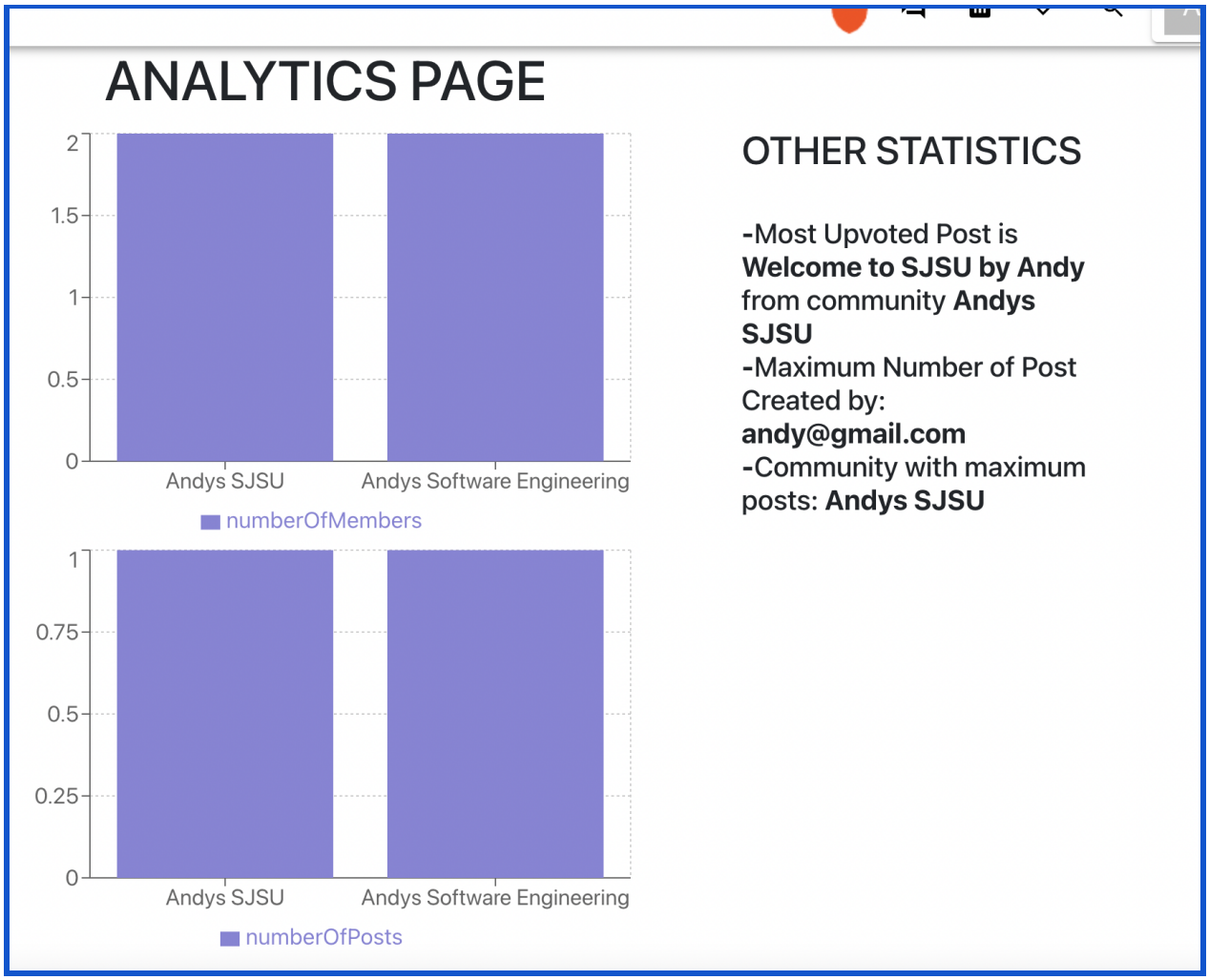

localhost:3000/communitymoderation
reddit
2
Your Community
Search Community Name
Andys Software Engineering
submitted on May 14, 2021 by andy@gmail.com
Approve
Invite
Andys SJSU
submitted on May 14, 2021 by andy@gmail.com
Approve
Invite
Next
Previous
flower2.jpeg
flower1.jpeg
Show All

Chat page

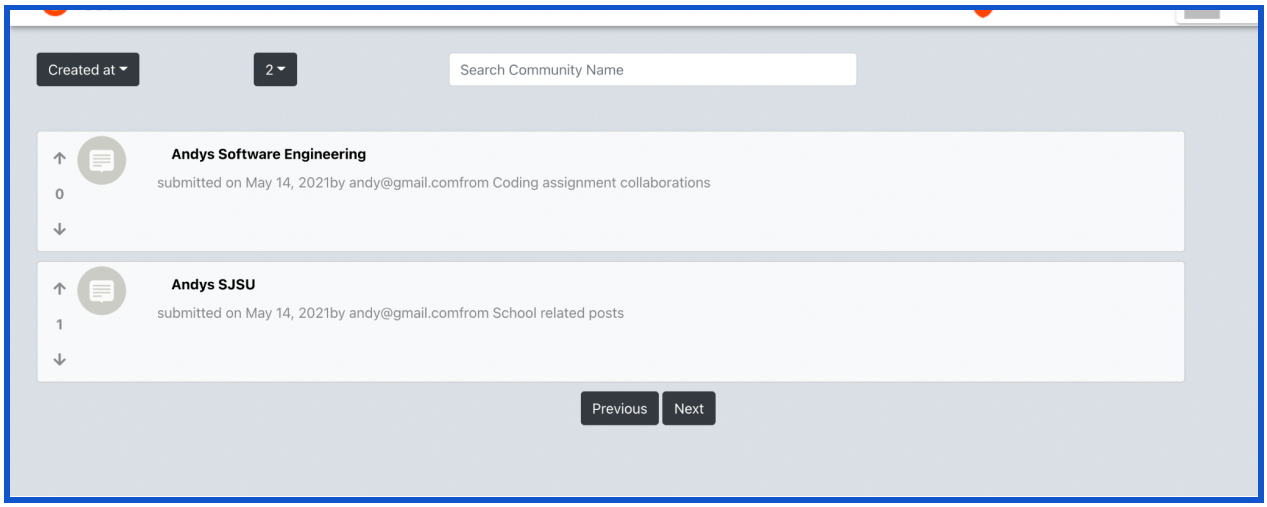

Start Chat
User Search
START CHAT



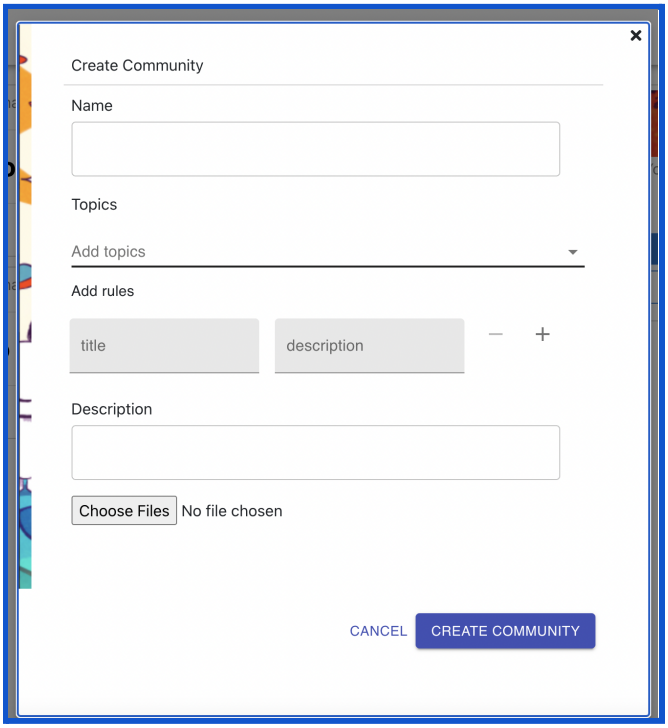
Analytics page



Search community page



Create Community



Code Listing of security and session objects:

```
"use strict";
var JwtStrategy = require("passport-jwt").Strategy;
var ExtractJwt = require("passport-jwt").ExtractJwt;
const passport = require("passport");
```

```
const { secret } = require("./config");
const pool = require("./mysqlConnection");

// Setup work and export for the JWT passport strategy
function auth() {
  var opts = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderWithScheme("jwt"),
    secretOrKey: secret,
  };
  passport.use(
    new JwtStrategy(opts, (jwt_payload, callback) => {
      const email = jwt_payload.email;
      const getUserQuery = "select name,email from User where email=?";

      pool.query(getUserQuery, [email], (err, sqlResult) => {
        if (err) {
          return callback(err, null);
        }
        if (sqlResult) {
          callback(null, sqlResult);
        } else {
          callback(null, false);
        }
      });
    })
  );
}
exports.auth = auth;
```

config files

```
"use strict";
const mysql = require("mysql");
const {
  mysqlHost,
  mysqlUser,
  mysqlPassowrd,
  myqlDatabase,
  mysqlPort,
```

```
} = require("./config");

const pool = mysql.createPool({
  connectionLimit: 100,
  host: mysqlHost,
  user: mysqlUser,
  port: mysqlPort,
  password: mysqlPassowrd,
  database: myqlDatabase,
});
```

Code Listing of Main Server Code:

```
app.use("/api", createCommunity);
app.use("/api", searchCommunity);
app.use("/api", addMessages);
app.use("/api", getMessages);
app.use("/api", createPost);
app.use("/api", getPost);
app.use("/api", requestedToJoinCommunity);
app.use("/api", inviteToJoinCommunity);
app.use("/api", acceptInvitationByUser);
app.use("/api", getCommunity);
app.use("/api", getProfile);
app.use("/api", updateProfile);
app.use("/api", imageUpload);
app.use("/api", acceptRequestToJoinCommunity);
app.use("/api", checkApprovedStatus);
app.use("/api", votingForPost);
app.use("/api", votingForCommunity);
app.use("/api", CommunitiesListByUser);
```

```
const mongoose = require("mongoose");
const Post = require("../model/Post");
const Community = require("../model/Community");
```

```
async function handle_request(msg, callback) {
  console.log("123");
  const createdByEmail = msg.createdByEmail;
  const communityName = msg.communityName;
  const postTitle = msg.postTitle;
  const flag = msg.flag;
  const images = msg.images;
  const url = msg.url;
  const text = msg.text;
  console.log("inside create abcd");
  console.log(flag);

  if (flag === "image") {
    const post = new Post({
      createdByEmail: createdByEmail,
      communityName: communityName,
      postTitle: postTitle,
      images: images,
      flag: flag,
    });

    console.log(post);
    await post.save(function (err, result) {
      if (err) {
        console.log(err);
      } else {
        Community.findOneAndUpdate(
          { communityName },
          { $inc: { numberOfPosts: 1 } },
          function (err, doc) {
            if (err) {
              callback(null, err);
            } else {
              callback(null, { message: "Post added" });
            }
          }
        );
      }
    });
```

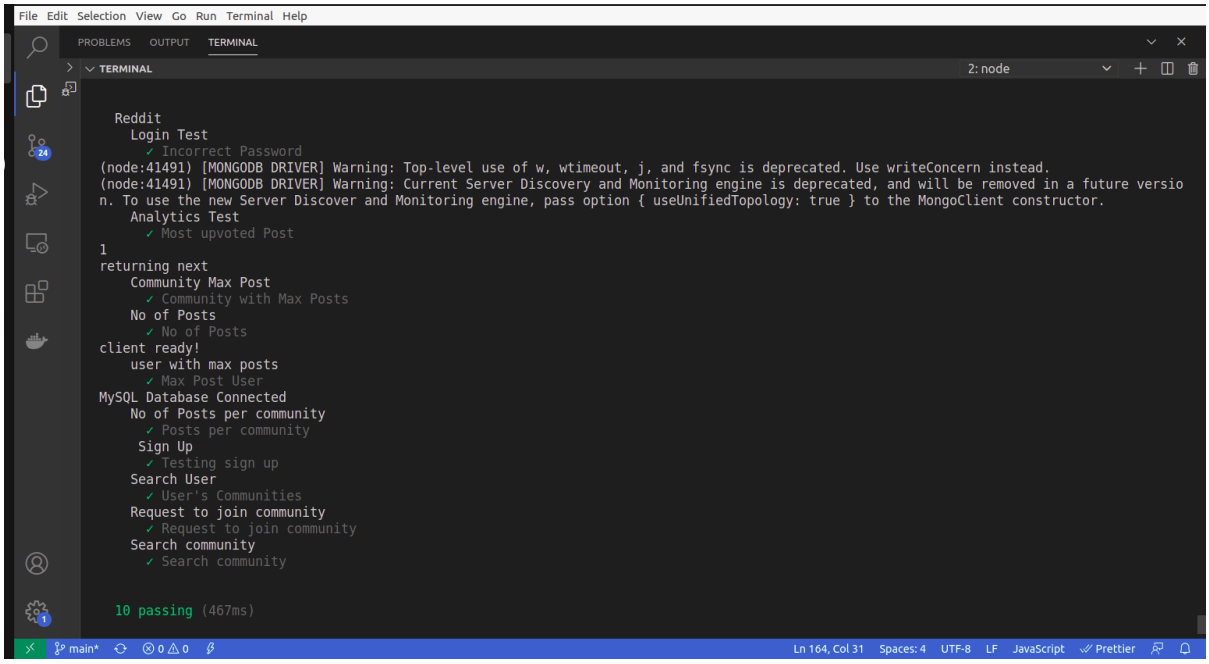
```
} else if (flag === "url") {
  const post = new Post({
    createdByEmail: createdByEmail,
    communityName: communityName,
    postTitle: postTitle,
    url: url,
    flag: flag,
  });

  console.log(post);
  await post.save(function (err, result) {
    if (err) {
      console.log(err);
    } else {
      Community.findOneAndUpdate(
        { communityName },
        { $inc: { numberOfPosts: 1 } },
        function (err, doc) {
          if (err) {
            callback(null, err);
          } else {
            callback(null, { message: "Post added" });
          }
        }
      );
    }
  });
} else if (flag === "text") {
  const post = new Post({
    createdByEmail: createdByEmail,
    communityName: communityName,
    postTitle: postTitle,
    text: text,
    flag: flag,
  });

  console.log(post);
  await post.save(function (err, result) {
    if (err) {
      console.log(err);
```

```
      } else {
        Community.findOneAndUpdate(
          { communityName },
          { $inc: { numberOfPosts: 1 } },
          function (err, doc) {
            if (err) {
              callback(null, err);
            } else {
              callback(null, { message: "Post added" });
            }
          }
        );
      }
    });
  }
}

exports.handle_request = handle_request;
```

Database Access and Connections:

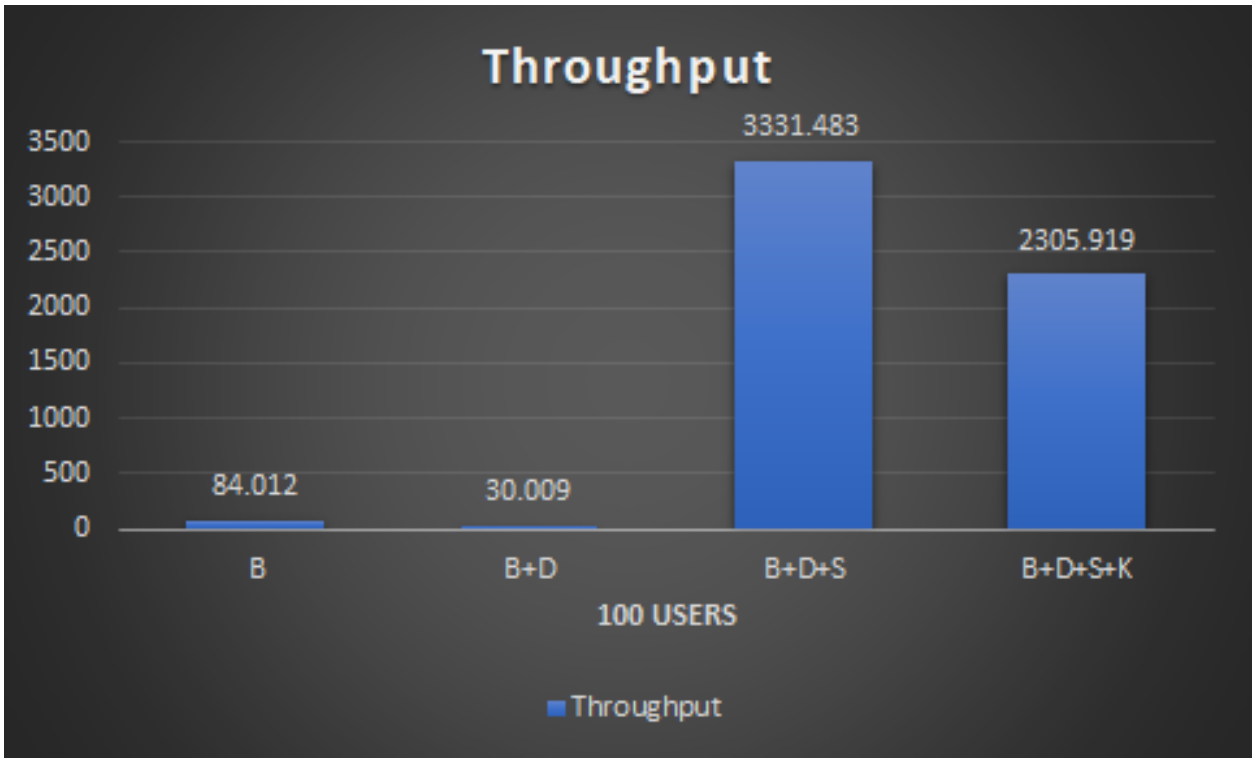
```
"use strict";
const mysql = require("mysql");
const {
  mysqlHost,
  mysqlUser,
  mysqlPassowrd,
  myqlDatabase,
  mysqlPort,
} = require("./config");

const pool = mysql.createPool({
  connectionLimit: 100,
  host: mysqlHost,
  user: mysqlUser,
  port: mysqlPort,
  password: mysqlPassowrd,
  database: myqlDatabase,
});

pool.getConnection((err) => {
  if (err) {
    throw "Error Occured: " + err;
  } else {
    console.log("MySQL Database Connected");
  }
});
module.exports = pool;
```

Mocha Test Screenshots:

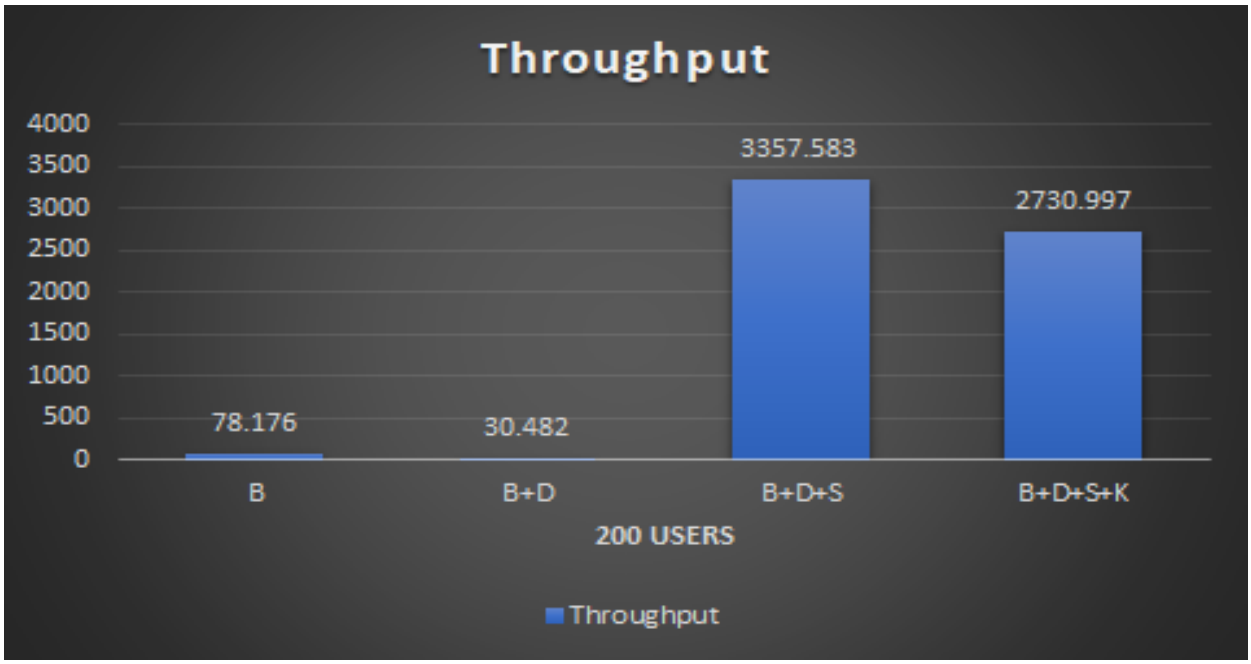


Performance Graphs:

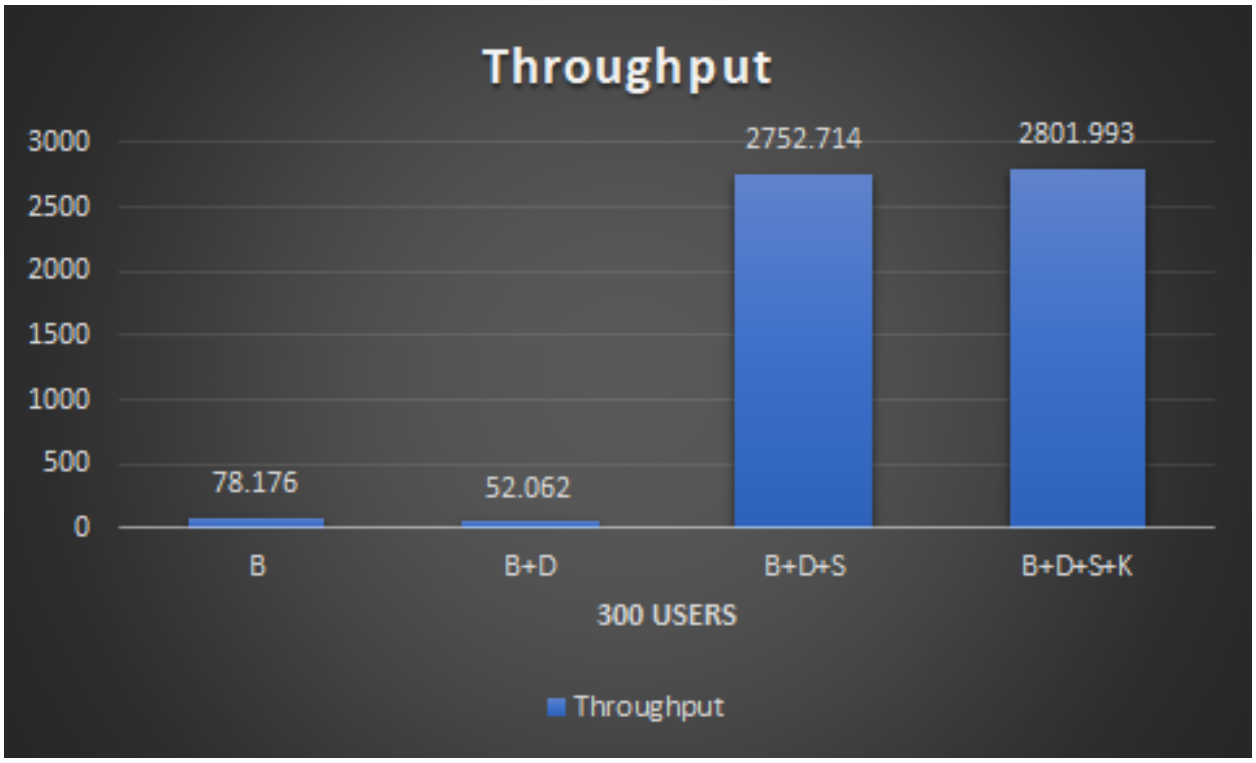
100 users :



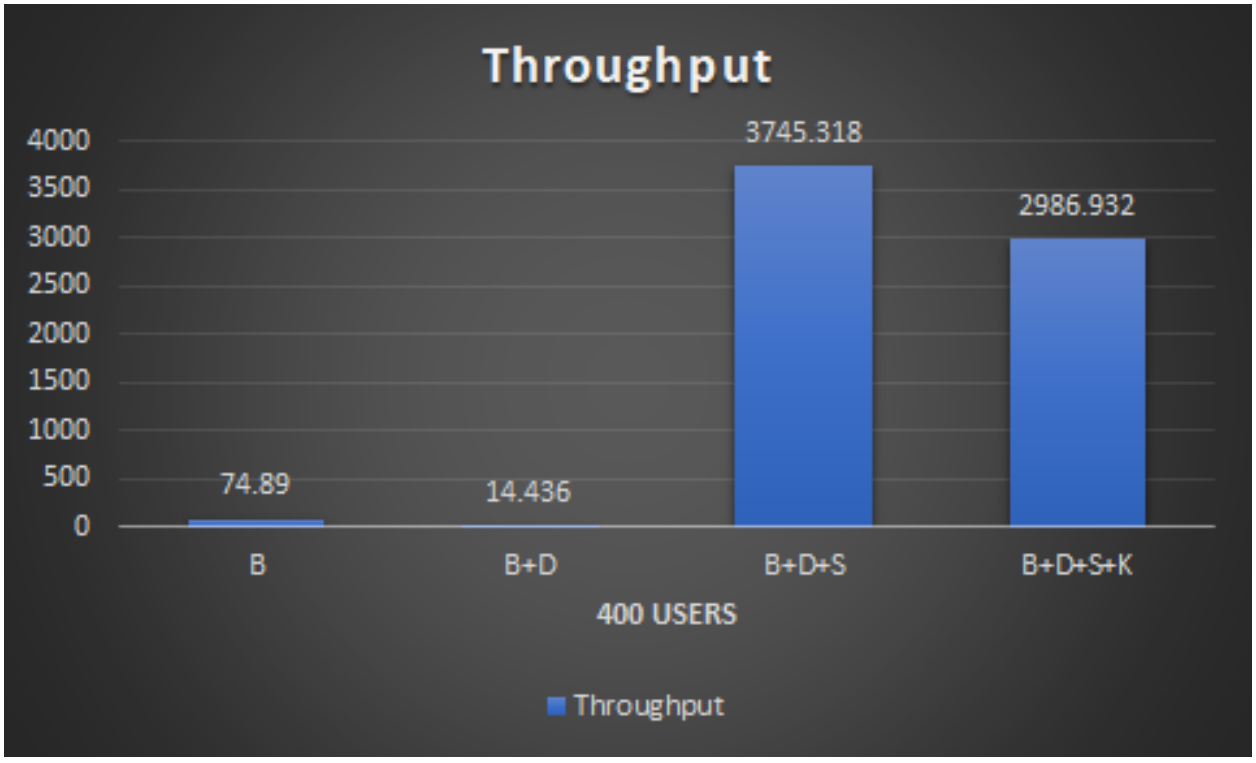
200 users:



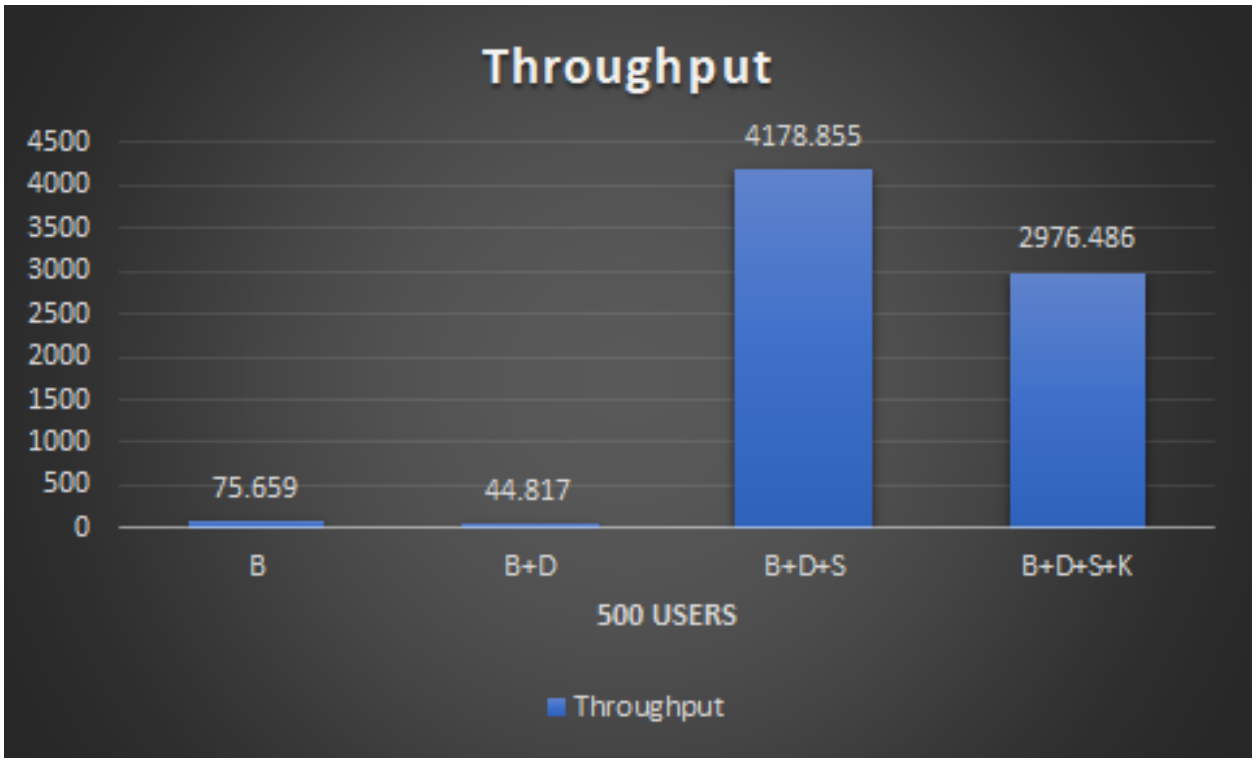
300 users:



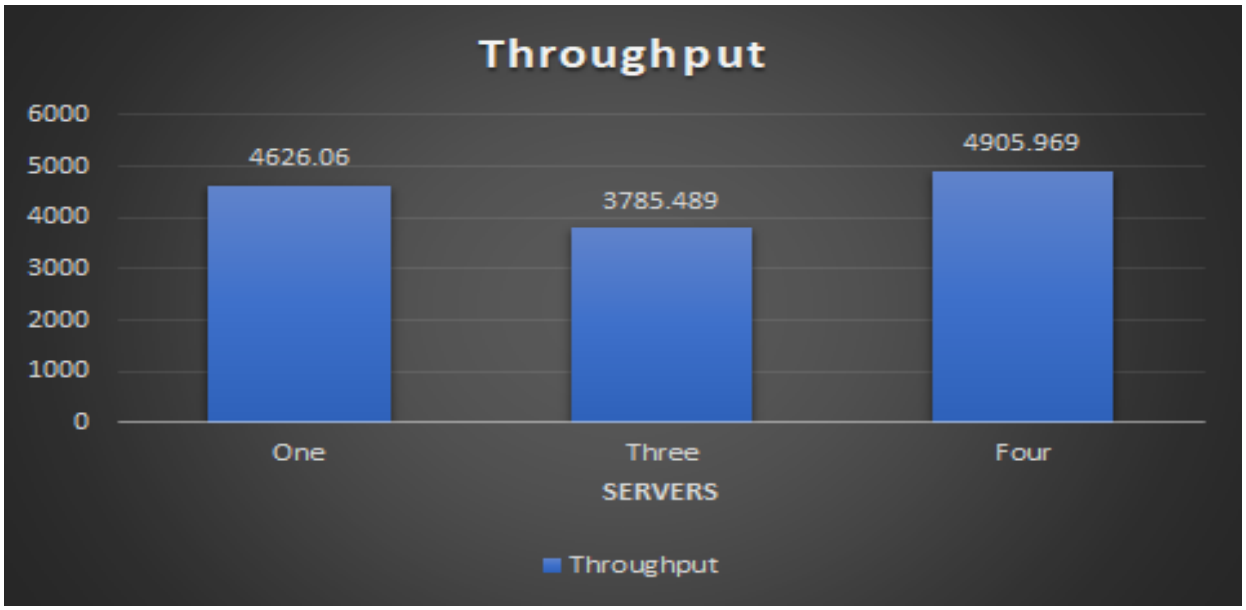
400 users:



500 users :



Performance on Load Balancing:



Performance Analysis and Comparison:

We populated the database with 10,000 messages to test various distributed system strategies. We have tested the getMessages API with the approaches mentioned in the document.

From the above graphs we can conclude the following

- There is a significant improvement in performance as soon as caching is introduced in the mix.
- Approaches with and without kafka (strategy 3&4) have small differences in performance. Although kafka impacts performance it provides benefits of scalability and high availability.

Code can be found at this github link:

https://github.com/SaiNikhilYandamuri/simulation-of-reddit